3D Point Cloud Generative Adversarial Network Based on Tree Structured Graph Convolutions

Dong Wook Shu*, Sung Woo Park*, and Junseok Kwon
School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea
seowok@naver.com pswkiki@gmail.com jskwon@cau.ac.kr

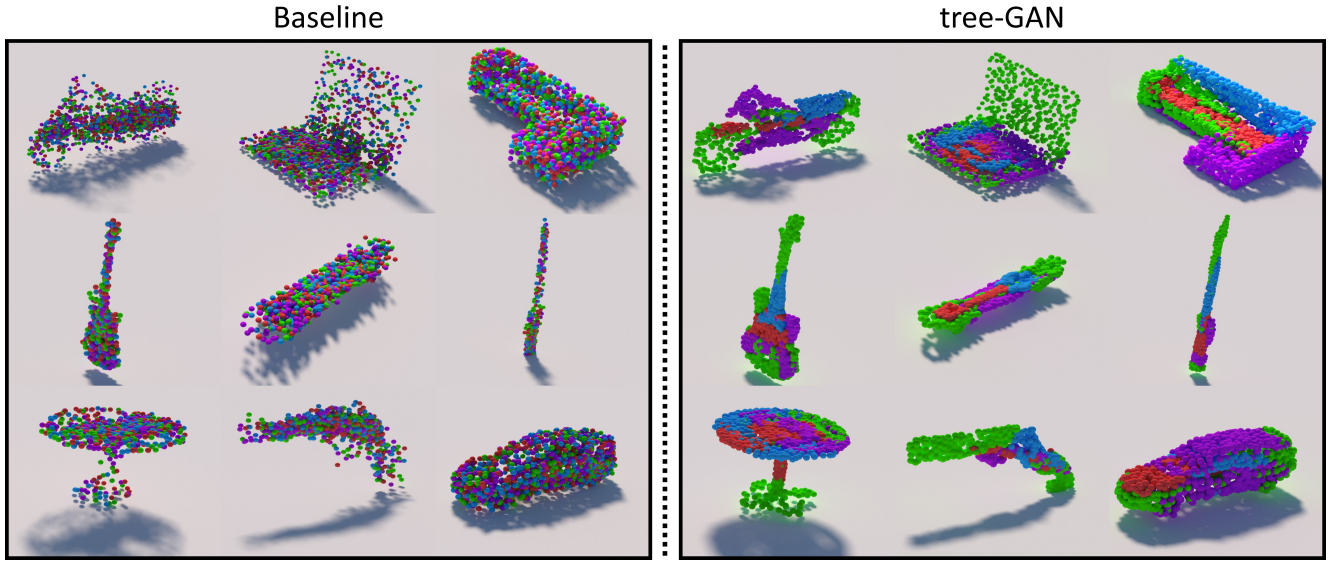

Figure 1. **Unsupervised 3D point clouds generated by our tree-GAN for multiple classes** (*e.g.*, Motorbike, Laptop, Sofa, Guitar, Skateboard, Knife, Table, Pistol, and Car from top-left to bottom-right). Our tree-GAN can generate more accurate point clouds than baseline (*i.e.*, r-GAN [1]), and can also produce point clouds for semantic parts of objects, which are denoted by different colors.

Abstract

In this paper, we propose a novel generative adversarial network (GAN) for 3D point clouds generation, which is called tree-GAN. To achieve state-of-the-art performance for multi-class 3D point cloud generation, a tree-structured graph convolution network (TreeGCN) is introduced as a generator for tree-GAN. Because TreeGCN performs graph convolutions within a tree, it can use ancestor information to boost the representation power for features. To evaluate GANs for 3D point clouds accurately, we develop a novel evaluation metric called Fréchet point cloud distance (FPD). Experimental results demonstrate that the proposed tree-GAN outperforms state-of-the-art GANs in terms of both conventional metrics and FPD, and can generate point clouds for different semantic parts without prior knowledge. The source code is available at https://github.com/seowok/TreeGAN.

1. Introduction

Recently, 3D data generation problems based on deep neural networks have attracted significant research interest and have been addressed through various approaches, including image-to-point cloud [11, 19], image-to-voxel [46], image-to-mesh [41], point cloud-to-voxel [6, 51], and point cloud-to-point cloud [48]. The generated 3D data has been used to achieve outstanding performance in a wide range of computer vision applications (*e.g.*, segmentation [30, 38, 45], volumetric shape representation [47], object detec-

*Authors contributed equally

tion [4, 35], feature extraction [24], contour detection [15], classification [29, 34], scene understanding [37, 44], and part semantic segmentation [22]).

However, little effort has been devoted to the development of generative adversarial networks (GANs) that can generate 3D point clouds in an unsupervised manner. To the best of our knowledge, the only works on GANs for transforming random latent codes (*i.e.*, z vectors) into 3D point clouds are [1] and [40]. The method in [1] generates point clouds using only fully connected layers. The method in [40] exploits local topology by using k-nearest neighbor techniques to produce geometrically accurate point clouds. However, it suffers from high computational complexity as the number of dynamic graph updates increases. Additionally, it can only generate a limited number of object categories (*e.g.*, chair, airplane, and sofa) using point clouds.

In this paper, we present a novel method called tree-GAN that can generate 3D point clouds from random latent codes in an unsupervised manner. It can also generate multi-class 3D point clouds without training on each class separately (*e.g.*, [40]). To achieve state-of-the-art performance in terms of both accuracy and computational efficiency, we propose a novel tree-structured graph convolution network (TreeGCN) as a generator for tree-GAN. The proposed TreeGCN preserves the ancestor information of each point and utilizes this information to extract new points via graph convolutions. A branching process and loop term with K supports in TreeGCN further enhance the representation power of points. These two properties enable TreeGCN to produce more accurate point clouds and express more diverse object categories. Additionally, we demonstrate that using the ancestors of features in TreeGCN is more efficient computationally than using the neighbors of features in traditional GCNs. Fig.1 shows the effectiveness of our tree-GAN.

The main contributions of this paper are fourfold.

• We present the novel tree-GAN method, which is a deep generative model that can generate multi-class 3D point clouds in unsupervised settings (Section 3).

• We introduce the TreeGCN based generator. The performance of traditional GCNs can be improved significantly by adopting the proposed tree structures for graph convolutions. Based on the proposed tree structures, tree-GAN can generate parts of objects by selecting particular ancestors (Section 4).

• We mathematically interpret the TreeGCN and highlight its desirable properties (Section 5).

• We present the Fréchet point cloud distance (FPD) metric to evaluate GANs for 3D point clouds. FPD can be considered as a nontrivial extension of Fréchet inception distance (FID) [17], which has been widely used for the evaluation of GANs (Section 6).

2. Related Work

Graph Convolutional Networks: Over the past few years, a number of works have focused on the generalization of deep neural networks for graph problems [3, 9, 16, 25]. Defferrard *et al.* [8] proposed fast-learning convolutional filters for graph classification problems. Using these filters, they significantly accelerated the spectral decomposition process, which was one of the main computational bottlenecks in traditional graph convolution problems with large datasets. Kipf and Welling [21] introduced scalable GCNs based on first-order approximations of spectral graph convolutions for semi-supervised classification, in which convolution filters only use the information from neighboring vertices instead of the information from the entire network.

Because the aforementioned GCNs were originally designed for classification problems, the connectivity of graphs was assumed to be given as prior knowledge. However, this setting is not appropriate for problems of dynamic model generation. For example, in unsupervised settings for 3D point cloud generation, the typologies of 3D point clouds are non-deterministic. Even for the same class (*e.g.*, chairs), 3D point clouds can be represented by various typologies. To represent the diverse typologies of 3D point clouds, our TreeGCN utilizes no prior knowledge regarding object models.

GANs for 3D Point Clouds Generation: GANs [13] for 2D image generation tasks have been widely studied with great success [10, 18, 23, 26, 33, 36, 42, 43, 49, 50], but GANs for 3D point cloud generation have rarely been studied in the computer vision field. Recently, Achlioptas *et al.* [1] proposed a GAN for 3D point clouds called r-GAN, generator of which is based on fully connected layers. As fully connected layers cannot maintain structural information, the r-GAN has difficulty in generating realistic shapes with diversity. Valsesia *et al.* [40] used graph convolutions for generators for GANs. At each layer of graph convolutions during training, adjacency matrices were dynamically constructed using the feature vectors from each vertex. Unlike traditional graph convolutions, the connectivity of a graph was not assumed to be given as prior knowledge. However, to extract the connectivity of a graph, computing the adjacency matrix at a single layer incurs quadratic computational complexity $O(V^2)$ where V indicates the number of vertices. Therefore, this approach is intractable for multi-batch and multi-layer networks.

Similar to the method in [40], our tree-GAN requires no prior knowledge regarding the connectivity of a graph. However, unlike the method in [40], the tree-GAN is computationally efficient because it does not construct adjacency matrices. Instead, the tree-GAN uses ancestor information from the tree to exploit the connectivity of a graph, in which only a list of tree structure from root node to leaf nodes is needed.

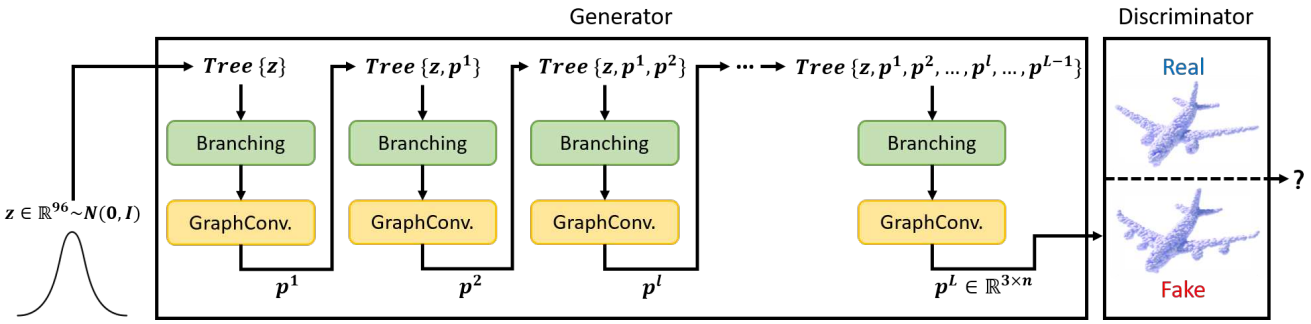

Figure 2. **Pipeline of the tree-GAN.** Our tree-GAN contains two networks, namely, discriminator (Section 3) and generator (Section 4). The generator takes a single point from a Gaussian distribution, $z \in R^{96}$, as an input. At each layer of the generator, GraphConv (Section 4.1) and Branching (Section 4.2) operations are performed to generate the l-th set of points, p^l. All points generated by previous layers are stored and appended to the tree of the current layer. The tree begins from the root node z, splits into child nodes via Branching operations, and modifies nodes by GraphConv operations. The generator produces 3D point clouds $x' = p^L \in R^{3\times n}$ as outputs, where p^L is the set of points at the final layer L and n is the total number of points. The discriminator differentiates between real and generated point clouds to force the generator to produce more realistic points. We use a discriminator similar to that in the r-GAN [1]. Please refer to supplementary materials for detailed network architectures.

Tree-structured Deep Networks: There have been several attempts to represent convolutional neural networks or long short-term memory using tree structures [5, 20, 27, 28, 31]. However, to the best of our knowledge, no previous methods have used tree structures for either graph convolutions or GANs. For example, Gadelha *et al.* [12] used tree-structured networks to generate 3D point clouds via variational autoencoder (VAE). However, this method needed the assumption that inputs are the 1D-ordered lists of points obtained by space-partitioning algorithms such as K-dimensional tree and random projection tree [7]. Thus, it required additional preprocessing steps for valid implementations. Because its network only comprised 1D convolution layers, the method could not extract the meaningful information from unordered 3D point clouds. In contrast, the proposed tree-GAN can not only deal with unordered points, but also extract semantic parts of objects.

3. 3D Point Cloud GAN

Fig.2 presents the pipeline of the proposed tree-GAN. To generate 3D point clouds x' from latent code z, we utilize the objective function introduced in Wasserstein GAN [2]. The loss function of a generator, L_{gen}, is defined as

$$L_{gen} = -\mathbb{E}_{z\sim\mathcal{Z}}[D(G(z))], \tag{1}$$

where G and D denote the generator and discriminator, respectively, and $\mathcal{Z}$ represents a latent code distribution. We design $\mathcal{Z}$ with a Normal distribution, $z \in \mathcal{N}(\mathbf{0}, I)$. The loss function of a discriminator, L_{disc}, is defined as

$$\begin{aligned} L_{disc} = \mathbb{E}_{z\sim\mathcal{Z}}[D(G(z))] - \mathbb{E}_{x\sim\mathcal{R}}[D(x)] \\ + \lambda_{gp}\mathbb{E}_{\hat{x}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2], \end{aligned} \tag{2}$$

where $\hat{x}$ are sampled from line segments between real and fake point clouds, $x' \sim G(z)$ and x denote generated and real point clouds, respectively, and $\mathcal{R}$ represents a real data distribution. In (2), we use a gradient penalty to satisfy the 1-Lipschitz condition [14], where λ_{gp} is a weighting parameter.

4. Proposed TreeGCN

To implement G in (1), we consider multi-layer graph convolutions with first-order approximations of the Chebyshev expansion introduced by [21] as follows:

$$p_i^{l+1} = \sigma\left(W^l p_i^l + \sum_{q_j^l \in N(p_i^l)} U^l q_j^l + b^l\right), \tag{3}$$

where $\sigma(\cdot)$ is the activation unit, p_i^l is the i-th node in the graph (*i.e.*, 3D coordinate of a point cloud) at the l-th layer, q_j^l is the j-th neighbor of p_i^l, and $N(p_i^l)$ is the set of all neighbors of p_i^l. Then, z and x' in (2) can be represented by $[p_1^0]$ and $[p_1^L\ p_2^L\ \cdots\ p_n^L]$, respectively, where L is the final layer and n is the number of points at L.

During training, GCNs find the best weights W^l and U^l and best bias b^l at each layer, then generate 3D coordinates for point clouds by using these parameters to ensure similarity to real point clouds. The first and second terms in (3) are called the *loop* and *neighbors* terms, respectively.

To enhance a conventional GCN such as that used in [21], we propose a novel GCN augmented with tree structures (*i.e.*, TreeGCN). The proposed TreeGCN introduces a tree structure for hierarchical GCNs by passing information from ancestors to descendants of vertices. The main unique characteristic of the TreeGCN is that each vertex updates its value by referring to the values of its ancestors in the tree instead of those of its neighbors. Traditional GCNs, such as those defined in (3), can be considered as methods that

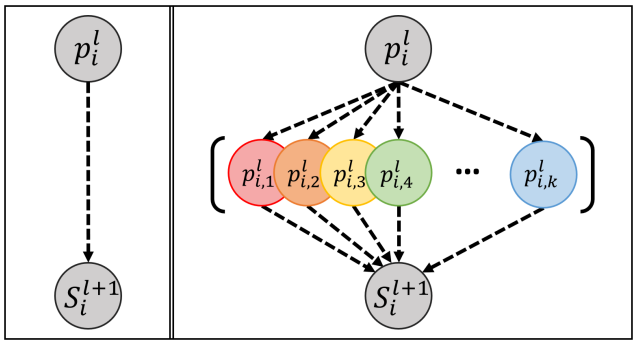

Figure 3. **Loop term with K-supports.** Left: a conventional loop term uses a single parameter W^l in (3) to learn the mapping from p_i^l to S_i^{l+1}. Right: our loop term introduces a fully connected layer with K nodes (*i.e.*, K supports, $p_{i,1}^l, \cdots, p_{i,k}^l$) to learn a more complex mapping from p_i^l to S_i^{l+1}.

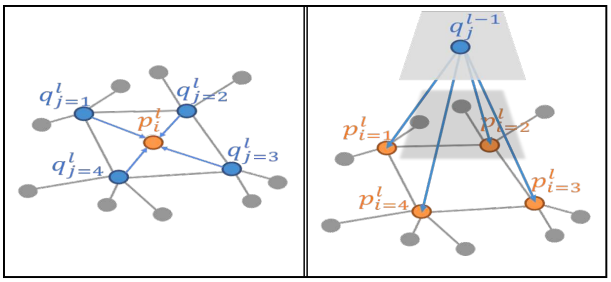

Figure 4. **Ancestor term**. Left: a conventional neighbor term uses neighbors of p_i^l (*e.g.*, $q_1^l, q_2^l, \cdots$) to generate p_i^{l+1}. Right: the proposed ancestor term uses ancestors of p_i^l (*e.g.*, $q_1^{l-1}, q_2^{l-2}, \cdots$) to generate p_i^{l+1}.

only refer to neighbors at a single depth. Then, the proposed graph convolution is defined as

$$p_i^{l+1} = \sigma\left(\mathbf{F}_K^l(p_i^l) + \sum_{q_j \in A(p_i^l)} U_j^l q_j + b^l\right), \quad (4)$$

where there are two major differences compared to (3). One is an improved conventional loop term using a subnetwork $\mathbf{F}_K^l$, where S_i^{l+1} is generated by K supports from $\mathbf{F}_K^l$. We call this term a *loop with K-supports*, as explained in Section 4.1. The other difference is the consideration of values from, all ancestors in the tree to update the value of a current point, where $A(p_i^l)$ denotes the set of all ancestors of p_i^l. We call this term *ancestors*, as explained in Section 4.1.

4.1. Advanced Graph Convolution

The proposed tree-structured graph convolution (*i.e.*, GraphConv in Fig.2) aims to modify the coordinates of points using the loop with K-supports and ancestor terms.

Loop term with K-supports: The goal of the new loop term in (4) is to propose the next point based on K supports instead of using only the single parameter W^l in (3) as follows:

$$S_i^{l+1} = \mathbf{F}_K^l(p_i^l), \quad (5)$$

where $\mathbf{F}_K^l$ is a fully connected layer containing K nodes. A conventional GCN using first-order approximations adopts a single parameter in its loop term to generate the next point from the current point. However, for large graphs, the representation capacity of a single parameter is insufficient for describing a complex point distribution. Therefore, our loop term utilizes K supports to represent a more complex distribution of points.

Ancestor term: For graph convolution, knowing the connectivity of a graph is very important because this information allows a GCN to propagate useful information from a vertex to other connected vertices. However, in our point cloud generation setting, it is impossible to use prior knowledge regarding connectivity because we must be able to generate diverse typologies of point clouds, even for the same object category. Therefore, the dynamic 3D point generation problem cannot be addressed using traditional GCNs because such networks assume that the connectivity of a graph is given. As a replacement for the neighbor term in (3), we define the *ancestor* in the second term of (4). This term combines all information from the ancestors q_j through a linear mapping U_j^l. Because each ancestor belongs to a different feature space at a different layer, our ancestor term can fuse all information from previous layers and different feature spaces. To generate the next point, the current point refers to its ancestors in various feature spaces to find the best mapping U_j^l to combine ancestor information effectively. By using this new *ancestor* term, our tree-GAN obtains several desirable mathematical properties, as explained in Section 5. Fig.4 illustrates the graph convolution process with the ancestor term.

4.2. Branching

Branching is a procedure for increasing the total number of points and is similar to up-sampling in 2D convolution. In branching, V_i^{l+1} transforms a single point $p_i^l \in R^3$ into d_l child points, where $[V_i^{l+1} \cdot p_i^l] \in R^{3 \times d_l}$. Therefore,

$$p_j^{l+1} = [V_i^{l+1} \cdot p_i^l]_j, \text{ for } j = 1, \cdots, d_l \quad (6)$$

where $[A]_j$ denotes the j-th column of matrix A. Then, the total number of points in the $(l+1)$-th layer is $|p^l| \times d_l$, where $|p^l|$ is the number of points in the l-th layer. In our experiments, we use different branching degrees for different layers (*e.g.*, $\{d_l\}_{l=1}^7 = \{1, 2, 2, 2, 2, 2, 64\}$). Note that the number of points in the final layer is $\prod_{l=1}^7 d_l = 2048$.

5. Mathematical Properties

In this section, we mathematically analyze the geometric relationships between generated points and demonstrate

how these relationships are formulated in the output Euclidean space via tree-structured graph convolutions.

Proposition 1. *Let p_s^L and p_d^L in (4) be generated points that share the same parents and different parents, respectively, with p_i^L in the final layer L. Let S_i^l in (5) be the loop of point p_i^l at the l-th layer. Hereafter, we omit the superscripts of p_s^l and S_i^l if the superscript l indicates the final layer L. Then,*

$$\|p_s - p_i\|^2 = \|S_s - S_i\|^2 , \tag{7}$$

and

$$\|p_d - p_i\|^2 \leq \sum_{l=1}^{L-1} \left\| S^l_{A_l(p_d)} - S^l_{A_l(p_i)} \right\|^2 + \|S_d - S_i\|^2 , \tag{8}$$

where $A_l(p)$ are all ancestors of point p in the l-th layer. For simplicity, we ignore the branch process and U_j^l in (5).

Based on Proposition 1, we can prove that following two statements are true:

• **The geometric distance between two points is determined by the number of shared ancestors.** If two points p_d and p_i have different ancestors, then the geometric distance between these points is calculated as the sum of differences between their ancestors in each layer l $\left(\textit{i.e.}, \ \left\| S^l_{A_l(p_d)} - S^l_{A_l(p_i)} \right\|^2 \text{ in (8)} \right)$ and the differences between their loops $\left(\textit{i.e.}, \ \|S_d - S_i\|^2 \text{ in (8)} \right)$. Thus, as their ancestors become increasingly different, the geometric distance between two points increases.

• **Geometrically related points share the same ancestors.** If two points p_s and p_i share the same ancestors, the geometric distance between these points is affected by only their *loop*s, as shown in (7). Thus, the geometric distance between points with the same ancestors in (7) can decreases compared to that between points with different ancestors in (8) as $\left\| S^l_{A_l(p_s)} - S^l_{A_l(p_i)} \right\|^2 = 0$.

Based on these two properties, our tree-GAN can generate semantic parts of objects, as shown in Fig.5, where points with the same ancestors are assumed to belong to the same parts of objects. We will explore this part generation problem for the proposed tree-GAN in Section 7.1.

6. Fréchet Point Cloud Distance

For quantitative comparisons between GANs, we require evaluation metrics that can accurately measure the quality of the 3D point clouds generated by GANs. In the case of 2D data generation problems, FID [17] is the most common metric. FID adopts pre-trained inception V3 models [39] to utilize their feature spaces for evaluation. Although the conventional metrics proposed by Achlioptas *et al.* [1] can be used to evaluate the quality of generated points by directly measuring matching distances between real and generated point clouds, they can be considered as sub-optimal metrics because the goal of a GAN is not to generate the similar samples (*e.g.*, MMD or CD) but to generate synthetic probability measures that are as close as possible to real probability measures. This perspective has been explored in unsupervised 2D image generation tasks using GANs [32, 17]. Therefore, we propose a novel evaluation metric for generated 3D point clouds called FPD.

Similar to FID, the proposed FPD calculates the 2-Wasserstein distance between real and fake Gaussian measures in the feature spaces extracted by PointNet [29] as follows:

$$\text{FPD}(\mathbb{P}, \mathbb{Q}) = \|\mathbf{m}_{\mathbb{P}} - \mathbf{m}_{\mathbb{Q}}\|_2^2 + \text{Tr}(\Sigma_{\mathbb{P}} + \Sigma_{\mathbb{Q}} - 2(\Sigma_{\mathbb{P}}\Sigma_{\mathbb{Q}})^{\frac{1}{2}}), \tag{9}$$

where $\mathbf{m}_{\mathbb{P}}$ and $\Sigma_{\mathbb{P}}$ are the mean vector and covariance matrix of the points calculated from real point clouds $\{x\}$, respectively, and $\mathbf{m}_{\mathbb{Q}}, \Sigma_{\mathbb{Q}}$ are the mean vector and covariance matrix calculated from generated point clouds $\{x'\}$, respectively, where $x \sim \mathbb{P}$ and $x' = G(z) \sim \mathbb{Q}$. In (9), $\text{Tr}(A)$ is the sum of the elements along the main diagonal of matrix A. In this paper, for evaluation purposes, we use both conventional evaluation metrics [1] and the proposed FPD.

7. Experimental Results

Implementation details: We used the Adam optimizer for both the generator and discriminator networks with a learning rate of $\alpha = 10^{-4}$ and other coefficients of $\beta_1 = 0$ and $\beta_2 = 0.99$. In generator, we used LeakyReLU as a nonlinearity function without batch normalization. The network architecture of discriminator was the same as that in r-GAN [1]. The gradient penalty coefficient was set to 10 and the discriminator was updated five times per iteration, while the generator was updated one time per iteration. As shown in Fig.2, a latent vector $z \in \mathbb{R}^{96}$ was sampled from a normal distribution $\mathcal{N}(0, I)$) to act as an input. Seven layers ($L = 7$) were used for the TreeGCN. The loop term of the TreeGCN in (5) had $K = 10$ supports. The total number of points in the final layer was set to $n = 2048$.

Comparison: There are only two conventional GANs for 3D point cloud generation: r-GAN [1] and the GAN proposed by Valsesia *et al.* [40]. Thus, the proposed tree-GAN was compared to these two GANs. While the conventional GANs in [40, 1] train separate networks for each class, our tree-GAN trains only a single network for multiple classes of objects.

Evaluation metrics: We evaluated the **tree-GAN** using ShapeNet[1], which is a large-scale dataset of 3D shapes, containing 16 object classes. Evaluations were conducted in terms of the proposed FPD (Section 6) and the metrics

[1] https://www.shapenet.org/

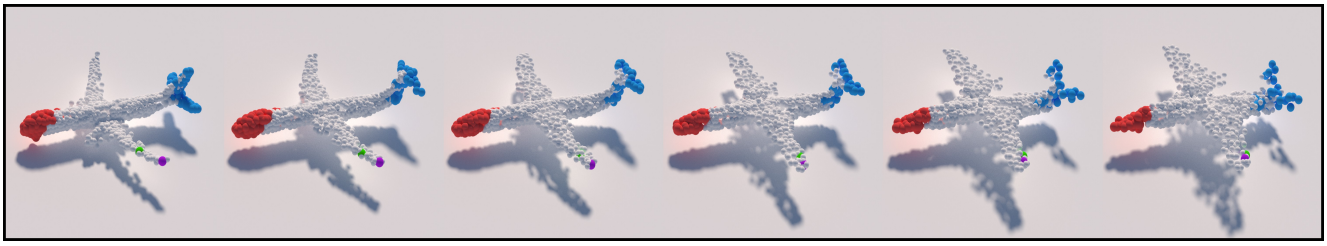

Figure 5. **Semantic part generation and interpolation results of our tree-GAN.** Red and blue point clouds are generated from different ancestors in the tree, which form geometrically different families of points. The leftmost and rightmost point clouds of the airplanes were generated from different noise inputs. The middle airplanes were obtained by interpolating between the leftmost and rightmost point clouds based on latent space representations.

used by Achlioptas et al [1]. As a reference model for FPD, we used the classification module of PointNet [29] because it can handle partial inputs of objects. This property is suitable for FPD because generated point clouds gradually form shapes, meaning point clouds can be partially complete during training. For the implementation of FPD, we first trained a classification module for 40 epochs to attain an accuracy of 98% for classification tasks. We then extracted a 1808-dimensional feature vector from the output of the dense layers to calculate the mean and covariance in (9).

7.1. Ablation Study

We analyze the proposed tree-GAN and examine its useful properties, namely unsupervised semantic part generation, latent space representation via interpolation, and branching.

Unsupervised semantic part generation: Our tree-GAN can generate point clouds for different semantic parts, even with no prior knowledge regarding those parts during training. The tree-GAN can perform this semantic generation owing to its tree-structured graph convolution, which is a unique characteristic among GAN-based 3D point cloud methods. As stated in Proposition 1, the geometric distance between points is determined by their ancestors in the tree. Different ancestors imply geometrically different families of points. Therefore, by selecting different ancestors, our tree-GAN can generate semantically different parts of point clouds. Note that these geometric families of points are consistent between different latent code inputs. For example, let $z_1, z_2 \sim \mathcal{N}(0, I)$ be sampled latent codes. Let $G(z_1) = [p_1\ p_2\ \dots\ p_{2048}]$ and $G(z_2) = [q_1\ q_2\ \dots\ q_{2048}]$ be their corresponding generated point clouds. Let J be a certain subset of 2048 point indices. Then, $G^J(z_1) = [p_j]_{j \in J}$ denotes the subset of $G(z_1)$ from the indices J. As shown in Fig. 5, if we select the same subsets of indices of point clouds, it results in the same semantic parts, even though the latent code inputs are different. For example, all red points indexed by J_h (*e.g.*, $G^{J_h}(z_1)$ and $G^{J_h}(z_2)$) represent the cockpits of airplanes, while all blue points indexed by J_t (*e.g.*, $G^{J_t}(z_1)$ and $G^{J_t}(z_2)$) represent the tails of airplanes.

From this ablation study, we can verify that the differences between ancestors determine the semantic differences between points and that two points with the same ancestor (*e.g.*, green and purple points in the left wings in Fig. 5) maintain their relative distances for different latent codes.

Interpolation: We interpolated 3D point clouds by setting the input latent code to $z_\alpha = (1-\alpha)z_1 + \alpha z_2$ based on six alphas $\alpha = [\alpha_1, \dots, \alpha_6]$. The leftmost and rightmost point clouds of the airplanes in Fig.5 were generated by $G(z_1)$ and $G(z_2)$, respectively. Our tree-GAN can also generate realistic interpolations between two point clouds.

Branching strategy: We conducted the experiments to show that the convergence dynamics of the proposed metric is not sensitive to different branching strategies. Like other experiments, the total number of the generated points are 2048 but different branching degrees were set (*e.g.*, $\{d_l\}_1^7 = \{1, 2, 2, 2, 2, 2, 64\}$, $\{1, 2, 4, 16, 4, 2, 2\}$, $\{1, 32, 4, 2, 2, 2, 2\}$). Please refer to convergence graphs in supplementary materials.

7.2. Comparisons with Other GANs

The proposed tree-GAN was quantitatively and qualitatively compared to other state-of-the-art GANs for point cloud generation, in terms of both accuracy and computational efficiency. Supplementary materials contain more results and comparisons for 3D point clouds generation.

Comparisons: Tables 1 and 2 contain quantitative comparisons in terms of the metrics used by Achlioptas *et al.* [1] (*i.e.*, JSD, MMD-CD, MMD-EMD, COV-CD, and COV-EMD) and the proposed FPD, respectively. The proposed tree-GAN consistently outperforms other GANs at a large margin in terms of all metrics, demonstrating the effectiveness of the proposed treeGCN.

For qualitative comparisons, we equally divided the entire index set into four subsets and painted the points in each subset with the same color. Although the real 3D point clouds were unordered as shown in Figs.1 and 6, our tree-GAN successfully generated 3D point clouds with intuitive semantic meaning without any prior knowledge, whereas r-

Table 1. **Quantitative comparison in terms of the metrics used by Achlioptas *et al.* [1].** Red and blue values denote the best and the second-best results, respectively. Because the original paper by Valsesia *et al.* [40] only presented point cloud results for chair and airplane classes, our tree-GAN was compared to [40] based on these classes. However, we additionally evaluated the proposed tree-GAN quantitatively for all 16 classes, as shown below. For networks with $\star$, we used results reported in [40]. Higher COV-CD and COV-EMD, and lower JSD, MMD-CD, and MMD-EMD indicate better methods.

Class	Model	JSD ↓	MMD-CD ↓	MMD-EMD ↓	COV-CD ↑	COV-EMD ↑
Chair	r-GAN (dense)*	0.238	**0.0029**	0.136	**33**	13
	r-GAN (conv)*	0.517	0.0030	0.223	23	4
	Valsesia *et al.* (no up.)*	**0.119**	0.0033	0.104	26	20
	Valsesia *et al.* (up.)*	**0.100**	**0.0029**	**0.097**	30	**26**
	tree-GAN (Ours)	**0.119**	**0.0016**	**0.101**	**58**	**30**
Airplane	r-GAN (dense)*	0.182	0.0009	0.094	**31**	9
	r-GAN (conv)*	0.350	**0.0008**	0.101	26	7
	Valsesia *et al.* (no up.)*	0.164	010010	0.102	24	13
	Valsesia *et al.* (up.)*	**0.083**	**0.0008**	**0.071**	**31**	**14**
	tree-GAN (Ours)	**0.097**	**0.0004**	**0.068**	**61**	**20**
All (16 classes)	r-GAN (dense)	0.171	0.0021	0.155	58	29
	tree-GAN (Ours)	**0.105**	**0.0018**	**0.107**	**66**	**39**

Table 2. **Quantitative comparison in terms of the proposed FPD.** The FPD for the real point clouds was almost nearly zero. This value can serve as the lower bound for the generated point clouds. Note that we could not evaluate the GAN proposed by Valsesia *et al.* [40] in terms of FPD because the source code was not available. Better methods have smaller values of FPD.

Class	Model	FPD ↓
Chair	r-GAN	1.860
	tree-GAN (Ours)	**0.809**
Airplane	r-GAN	1.016
	tree-GAN (Ours)	**0.439**
All (16 classes)	r-GAN	4.726
	tree-GAN (Ours)	**3.600**
	Real (Low bound)	**0**

GAN failed to generate semantically ordered point clouds. Additionally, our tree-GAN could generate detailed and complex parts of objects, whereas r-GAN generated more dispersed point distributions. Fig.6 presents qualitative results of our tree-GAN. The tree-GAN generated realistic point clouds for multi-object categories and produced very diverse typologies of point clouds for each class.

Computational cost: In methods using *static links* for graph convolution, adjacency matrices are typically used for the convolution of vertices. Although these methods are known to produce good results for graph data, prior knowledge regarding connectivity is required. In other methods that use *dynamic links* for graph convolution, adjacency matrices must be constructed from vertices to derive connectivity information for every convolution layer instead of using prior knowledge. For example, let L, B, V_l denote the number of layers, batch size, and induced vertex size of an output graph at the l-th layer, respectively. The methods described above require additional computations to utilize connectivity information. These computations require time and memory resources on the order of $\sum_{l=1}^{L} B \times V_l \times V_l$. However, our TreeGCN does not require any prior connectivity information like *static link* methods and does not require additional computation like *dynamic link* methods. Therefore, our network can use time and memory resources much more efficiently and requires less resources on the order of $\sum_{l=1}^{L} B \times V_l$.

8. Conclusion

In this paper, we proposed a generative adversarial network called the tree-GAN that can generate 3D point clouds in an unsupervised manner. The proposed generator for tree-GAN, which is called tree-GCN, preforms graph convolutions based on tree structures. The tree-GCN utilizes ancestor information from a tree and employs multiple supports to represent 3D point clouds. Thus, the proposed tree-GAN outperforms other GAN based point cloud generation methods in terms of accuracy and computational efficiency. Through various experiments, we demonstrated that the tree-GAN can generate semantic parts of objects without any prior knowledge and can represent 3D point clouds in latent spaces via interpolation.

9. Acknowledgement

This work was supported by Institute for Information & communications Technology Planning & evaluation(IITP) grant funded by the Korea government(MSIT) (No.2017-0-01780).

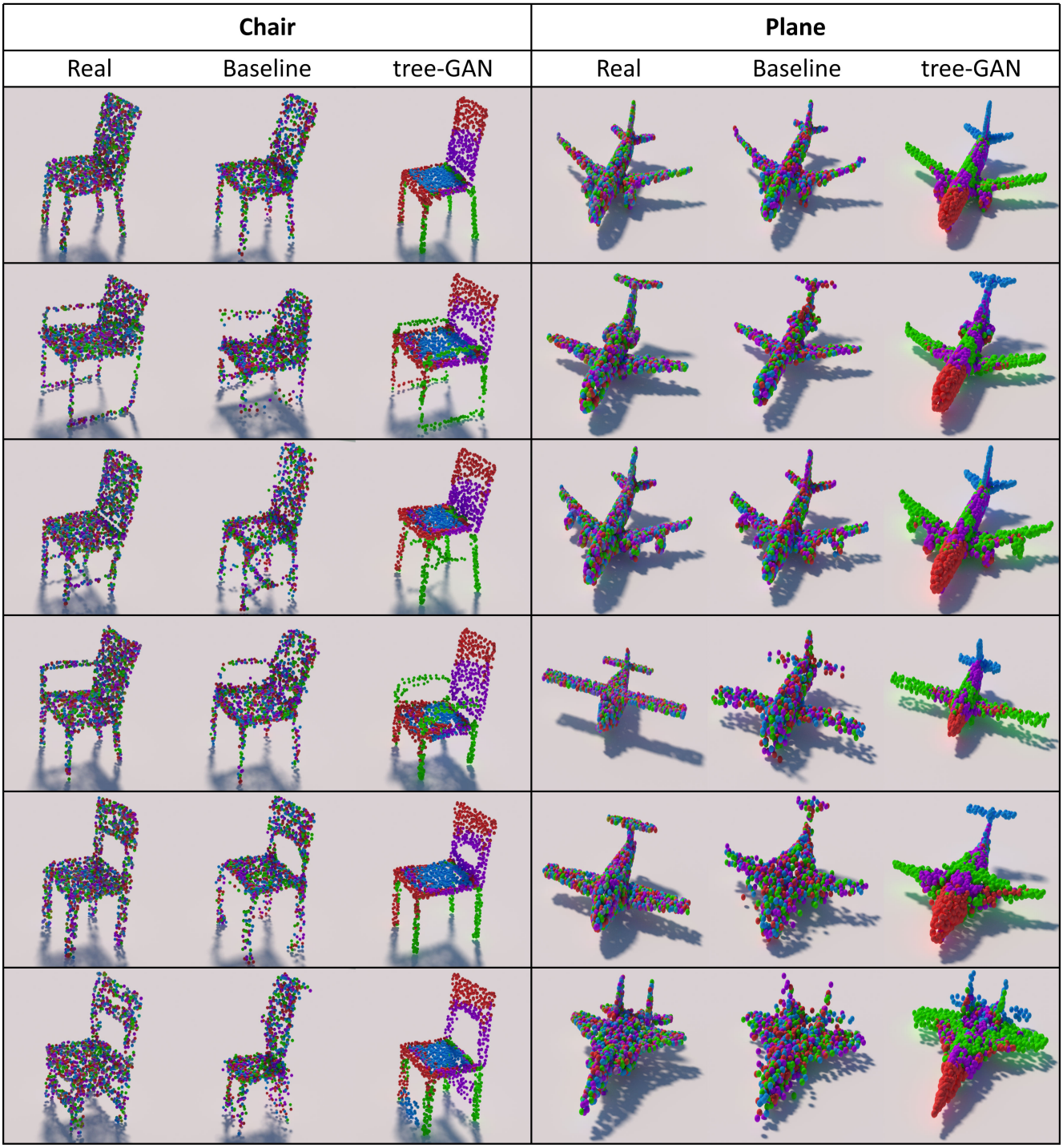

Figure 6. **Unsupervised 3D point cloud generation results of baseline (*i.e.* r-GAN [1]) and our tree-GAN.** The proposed tree-GAN generates more accurate and detailed point clouds of objects as comparison with r-GAN, and produces point clouds for each part of the objects even with no prior knowledge on that part. The point clouds generated by the tree-GAN can represent a variety of geometrical typologies for each class. The first, second, and third columns show point clouds of ground truth, baseline, and tree-GAN, respectively.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3D point clouds. In *ICLR*, 2018. 1, 2, 3, 5, 6, 7, 8

[2] Martin Arjovsky, Soumith Chintala, and Leon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017. 3

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014. 2

[4] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G. Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3D object proposals for accurate object class detection. In *NIPS*, 2015. 2

[5] Zhou Cheng, Chun Yuan, Jiancheng Li, and Haiqin Yang. TreeNet: Learning sentence representations with unconstrained tree structure. In *IJCAI*, 2018. 3

[6] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Niener. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR*, 2017. 1

[7] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *STOC*, 2008. 3

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016. 2

[9] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2015. 2

[10] Kiana Ehsani, Roozbeh Mottaghi, and Ali Farhadi. SeGAN: Segmenting and generating the invisible. In *CVPR*, 2018. 2

[11] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3D object reconstruction from a single image. In *CVPR*, 2017. 1

[12] Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution tree networks for 3d point cloud processing. In *ECCV*, 2018. 3

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 2

[14] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein GANs. In *NIPS*, 2017. 3

[15] Timo Hackel, Jan D. Wegner, and Konrad Schindler. Contour detection in unstructured 3d point clouds. In *CVPR*, 2016. 2

[16] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015. 2

[17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017. 2, 5

[18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial nets. In *CVPR*, 2017. 2

[19] Li Jiang, Shaoshuai Shi, Xiaojuan Qi, and Jiaya Jia. GAL: Geometric adversarial loss for single-view 3D-object reconstruction. In *ECCV*, 2018. 1

[20] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization. In *ICML*, 2017. 3

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 2, 3

[22] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *ICCV*, 2017. 2

[23] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017. 2

[24] Yangyan Li, Soeren Pirk, Hao Su, Charles R. Qi, and Leonidas J. Guibas. FPNN: Field probing neural networks for 3D data. In *NIPS*, 2016. 2

[25] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016. 2

[26] Jianxin Lin, Yingce Xia, Tao Qin, Zhibo Chen, and Tie-Yan Liu. Conditional image-to-image translation. In *CVPR*, 2018. 2

[27] Calvin Murdock, Zhen Li, Howard Zhou, and Tom Duerig. Blockout: Dynamic model selection for hierarchical deep networks. In *CVPR*, 2016. 3

[28] Hyeonseob Nam, Mooyeol Baek, and Bohyung Han. Modeling and propagating CNNs in a tree structure for visual tracking. *arXiv preprint arXiv:1608.07242*, 2016. 3

[29] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, 2017. 2, 5, 6

[30] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 1

[31] Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. Tree-CNN: A deep convolutional neural network for lifelong learning. *arXiv preprint arXiv:1802.05800*, 2018. 3

[32] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training GANs. In *NIPS*. 2016. 5

[33] Xinchen Yan Lajanugen Logeswaran Bernt Schiele Scott Reed, Zeynep Akata and Honglak Lee. Generative adversarial text to image synthesis. In *ICML*, 2016. 2

[34] Richard Socher, Brody Huval, Bharath Bath, Christopher D. Manning, and Andrew Y. Ng. Convolutional-recursive deep learning for 3D object classification. In *NIPS*, 2012. 2

[35] Shuran Song and Jianxiong Xiao. Deep sliding shapes for a modal 3D object detection in RGB-D images. In *CVPR*, 2016. 2

[36] Yibing Song, Chao Ma, Xiaohe Wu, Lijun Gong, Linchao Bao, Wangmeng Zuo, Chunhua Shen, Rynson Lau, and Ming-Hsuan Yang. VITAL: Visual tracking via adversarial learning. In *CVPR*, 2018. 2

[37] Byung soo Kim, Pushmeet Kohli, and Silvio Savarese. 3D scene understanding by voxel-CRF. In *CVPR*, 2013. 2

[38] Simon Christoph Stein, Markus Schoeler, Jeremie Papon, and Florentin Worgotter. Object partitioning using local convexity. In *CVPR*, 2014. 1

[39] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 5

[40] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3D point clouds via graph convolution. In *ICLR*, 2019. 2, 5, 7

[41] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2Mesh: Generating 3d mesh models from single RGB images. In *ECCV*, 2018. 1

[42] Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. In *ECCV*, 2016. 2

[43] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-Fast-RCNN: Hard positive generation via adversary for object detection. In *CVPR*, 2017. 2

[44] Yan Wang, Rongrong Ji, and Shih-Fu Chang. Label propagation from imageNet to 3D point clouds. In *CVPR*, 2013. 2

[45] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. 1

[46] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NIPS*, 2016. 1

[47] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D shapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015. 1

[48] Yaoqing Yang, Ceng, Yiru Shen, and Dong Tian. FoldingNet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, 2018. 1

[49] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative image inpainting with contextual attention. In *CVPR*, 2018. 2

[50] Zizhao Zhang, Lin Yang, and Yefeng Zheng. Translating and segmenting multimodal medical volumes with cycle- and shape-consistency generative adversarial network. In *CVPR*, 2018. 2

[51] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-end learning for point cloud based 3D object detection. In *CVPR*, 2018. 1